Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 310 (2015) 27–47
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Fluid Performability Analysis of Nested Automata Models

Luca Bortolussi

Department of Mathematics and Geosciences, University of Trieste, ISTI-CNR Pisa, luca@dmi.units.it

Jane Hillston

School of Informatics, University of Edinburgh, jane.hillston@ed.ac.uk

Mirco Tribastone

Electronics and Computer Science, University of Southampton, m.tribastone@soton.ac.uk

Abstract

In this paper we present a class of nested automata for the modelling of performance, availability, and reliability of software systems with hierarchical structure, which we call *systems of systems*. Quantitative modelling provides valuable insight into the dynamic behaviour of software systems, allowing non-functional properties such as performance, dependability and availability to be assessed. However, the complexity of many systems challenges the feasibility of this approach as the required mathematical models grow too large to afford computationally efficient solution. In recent years it has been found that in some cases a fluid, or mean field, approximation can provide very good estimates whilst dramatically reducing the computational cost.
The systems of systems which we propose are hierarchically arranged automata in which influence may be exerted between siblings, between parents and children, and even from children to parents, allowing a wide range of complex dynamics to be captured. We show that, under mild conditions, systems of systems can be equipped with fluid approximation models which are several orders of magnitude more efficient to run than explicit state representations, whilst providing excellent estimates of performability measures. This is a significant extension of previous fluid approximation results, with valuable applications for software performance modelling.

Keywords: Systems of systems, fluid approximation, software performance modelling.

1 Introduction

Modelling and analysis of nested (or hierarchical) structures occurs frequently in many domains, including for example, software systems and biological processes. For instance in the software context, a cloud environment may be thought of as a collection of many computers, each containing other components, such as processors and threads [10]. Thus there are three levels: processes and threads, computers and the cloud itself, and at each of these levels the behaviour of an individual

http://dx.doi.org/10.1016/j.entcs.2014.12.011

entity can be described by an automaton. Note that the hierarchical nesting in these systems is genuinely *structural* and not an abstraction used to hide detail as is done, for example, in UML state machines. The organisational complexity of these systems means that it is hard to predict their behaviour and it is imperative that performance and reliability characteristics are investigated prior to deployment. When formal reasoning about these systems is performed with reactive models based on a discrete state-space representation, the problem size grows extremely quickly with the population of components, making the analysis infeasible in practice.

In this paper, we present a Markov model of nested automata which we call *system of systems*. Automata are hierarchically organised in a tree (Fig.1 a)); the behaviour of an automaton can be affected by the state of its siblings (*horizontal interaction*; Fig.1 c)). Each automaton contains other automata, and the dynamics of a parent may have an impact on that of its children, and vice versa (*vertical interaction*; Fig.1 d)). Each node of the tree is assigned a multiplicity, which indicates how many copies of the stochastic automaton are present in the system of systems *within each copy* of its parent automaton (Fig.1 b)).

Nested Markov models have previously been proposed as a good model for hierarchically organised software systems, but limited progress has been made against the heavy computational costs due to layering and large multiplicities. Previously proposed techniques seek to exploit symmetries but still yield a Markov chain. For example, Buchholz exploits Kronecker algebra with hierarchical models that can express certain classes of stochastic Petri nets and queueing networks, but his work is limited to only two levels of nesting [4,5]. Lanus and Trivedi consider a class of hierarchical Markov models where the states of automata of arbitrary size can be partitioned in such a way that a reduced model can be constructed which preserves steady-state reward measures of availability and performance [11]. However, the state space size of the CTMC is still dependent (exponentially in the worst case) on the number of so-reduced automata.

In this paper, we introduce a mean-field approximation based on a system of ordinary differential equations (ODEs) which initially associates an equation with each element of the CTMC state descriptor, estimating the expectation. Unfortunately, the state descriptor grows exponentially with the number of levels and polynomially with the branching level in the class tree, the number of automata in each class, and the number of states of each automata class. This clearly hinders even the applicability of fluid models for nested systems of moderate size. To address this issue, we exploit a property of symmetry between such equations which, informally, shows that two distinct equations for any two automata copies of the same kind (i.e., belonging to the same node in the tree) yield the same ODE solution. Thus, a significant reduction of the ODE system size is possible by considering a representative set of equations for each node in the tree, independently from all the multiplicities involved.

The basic result of ODE symmetry is analogous to [15], which establishes a form of ODE lumpability in the context of the Markovian process algebra PEPA [9]. In this respect, this paper represents a significant improvement owing to its generality

and much wider scope of applicability. Specifically, in [15] horizontal interaction is restricted to the semantics of PEPA. This excludes, for instance, the possibility of studying nested systems with more general forms of interactions, such as the law of mass action used in certain networking models [16]. Instead, our results do not depend on the actual laws of interaction used, provided they yield an ODE system with a unique solution. With respect to vertical interaction, this paper relaxes the restrictions imposed by the choice of a specific synchronisation operator in process algebra. For instance, in PEPA (and in CSP-based calculi in general), vertical interaction can be modelled by considering a shared action, α, between a parent process P and its children, $C_1, \ldots, C_n$, in a composite process: $P \parallel_{\{\alpha\}} (C_1 \parallel_L \ldots \parallel_L C_n)$. In this case, if $\alpha \in L$ then the semantics enforces that an α-action is witnessed only when *all processes* can perform it. On the other hand, if $\alpha \notin L$ *only one* of the processes C_i will perform an action in synchronisation with its parent. In CCS and other process calculi based on complementary actions the situation is even more restrictive, even if transactions on binary interactions are introduced. In our modelling formalism, vertical interaction is obtained by introducing the notion of *causal map*. Using a causal map the modeller may specify, for instance, which states of the child automata are susceptible to an action performed by the parent, and with what probability a child changes its susceptible state when that action is witnessed. It can be shown that this level of expressiveness cannot be obtained by using PEPA or other available Markovian process algebra (e.g., [1,8]).

Paper Overview

Section 2 presents our Markov model of systems of systems, helped by a simple running example that illustrates the main definitions. Section 3 discusses the fluid approximation and presents the result of ODE symmetry reduction. Section 4 uses a case study of a performability model for a hierarchical distributed computing system, for the purposes of an extensive numerical validation which considers the accuracy of the fluid approximation and its computational advantage over stochastic analysis. Finally, Section 5 concludes the paper.

2 Nested Automata

A *system of systems* is a hierarchical model consisting of Markov automata which *contain* other automata, with an arbitrary level of nesting. Here we are interested in systems in which at any level of their hierarchical organisation consist of a population of interacting agents. Examples of this sort of systems are ubiquitous: In biology, tissues are composed of many cells, each containing many different biochemical molecules interacting together. Server farms contain many computers, each running a potentially large number of processes. In this latter case, the Markov automata at the higher level may represent server farms, and they will contain Markov automata modelling the computers inside the farms, each of which contains a population of automata representing the running processes. This notion of hierarchical containment is illustrated by means of an example in Figure 1, using

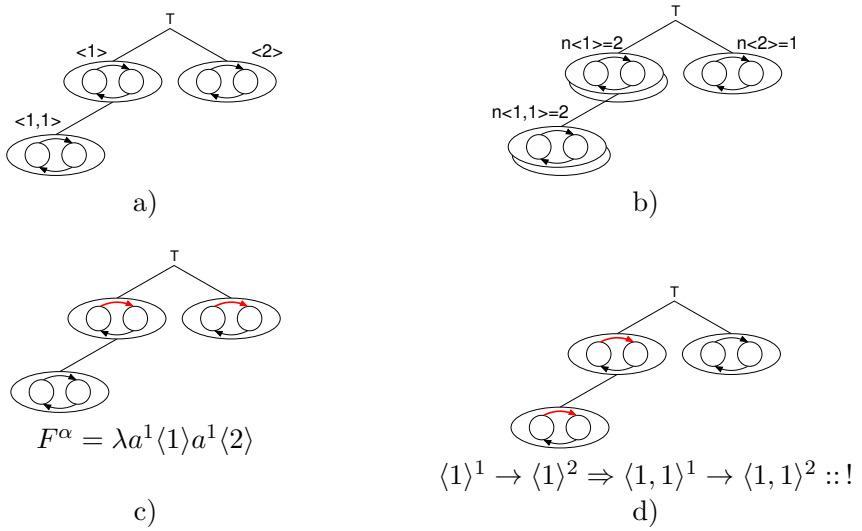

Fig. 1. a) A system of systems $\mathcal{A}$; b) an instance of a system of systems; c) horizontal interaction; d) vertical interaction.

the notation that will be introduced in the remainder of this section.

We now turn to describe the structure of a system of systems model. The idea is that we will describe at each level of the hierarchy a prototype agent for each type of agent populating that level. This is captured in the notion of *agent class*. Then, such classes will be instantiated, specifying how many agent instances there are for each class in the system.

2.1 Structure

Let $\mathcal{A}$ be the set of all automata classes of a model. We can think of each class as a type of element or agent within the system. So, for example, in a server farm, farms, computers and jobs, may all be distinct automata classes within the system description. A system of systems is specified by a tree with nodes $\mathcal{A}$ describing the hierarchical organisation of such classes. Thus in the server farm example, a computer will be the child of a farm and be parent to several jobs.

In order to talk about the different agent classes within the tree, we need some notation for their coordinates. For this, we assume a fixed and well-defined visit strategy (for instance, depth-first) and denote by D the tree depth. We let $\langle i_1, i_2, \ldots, i_l\rangle \in \mathcal{A}$ denote the automata class at level l of the tree which is reached by navigating the tree starting from the i_1-th automata class below the root, then taking its i_2-th child, and so on. Note that we do not require balanced trees and so, for instance, $\langle 1,1,1\rangle$ may belong to the tree but $\langle 2,1\rangle$ may not. In the notation we sometimes abbreviate $\langle i_1, i_2, \ldots, i_l\rangle$ to $\langle \boldsymbol{i}_l\rangle$, where $\boldsymbol{i}_l$ is a vector of indices of length l, or similarly, to $\langle \boldsymbol{i}_{l-1}, i_l\rangle$. We use $c\langle \boldsymbol{i}_l\rangle$ to denote the number of children of $\langle \boldsymbol{i}_l\rangle$. These will be indexed by $\langle \boldsymbol{i}_l, 1\rangle, \langle \boldsymbol{i}_l, 2\rangle, \ldots, \langle \boldsymbol{i}_l, c\langle \boldsymbol{i}_l\rangle\rangle$.

Having established the hierarchical organisation of automata classes we now turn to describe automata classes themselves. Essentially, each automaton will be a finite state machine, which will change state probabilistically (at random times),

due to interactions with other automata specified by system level rules. We start by providing the description of the automata structure: An automata class $\langle \boldsymbol{i}_l \rangle$ is defined by the tuple

$$\langle \boldsymbol{i}_l \rangle = \left(\Sigma_{\langle \boldsymbol{i}_l \rangle}, \longrightarrow_{\langle \boldsymbol{i}_l \rangle}, n\langle \boldsymbol{i}_l \rangle \right),$$

where

- $\Sigma_{\langle \boldsymbol{i}_l \rangle}$ is the automaton's state space, with states denoted by $\langle \boldsymbol{i}_l \rangle^j$, with $1 \leq j \leq d\langle \boldsymbol{i}_l \rangle$, where $d\langle \boldsymbol{i}_l \rangle = |\Sigma_{\langle \boldsymbol{i}_l \rangle}|$.
- $\longrightarrow_{\langle \boldsymbol{i}_l \rangle} \subseteq \Sigma_{\langle \boldsymbol{i}_l \rangle} \times \Sigma_{\langle \boldsymbol{i}_l \rangle}$ is the set of transitions between states. When the context is clear we abbreviate $\longrightarrow_{\langle \boldsymbol{i}_l \rangle}$ by $\longrightarrow$. Furthermore, we use the typical notation $\langle \boldsymbol{i}_l \rangle^j \to \langle \boldsymbol{i}_l \rangle^{j'}$ if $\left(\langle \boldsymbol{i}_l \rangle^j, \langle \boldsymbol{i}_l \rangle^{j'} \right) \in \longrightarrow_{\langle \boldsymbol{i}_l \rangle}$.
- $n\langle \boldsymbol{i}_l \rangle \in \mathbb{N}$ is the population size, i.e. it specifies how many distinct copies of the automaton $\langle i_1, i_2, \ldots, i_l \rangle$ are present *within each copy* of its automaton parent, $\langle i_1, i_2, \ldots, i_{l-1} \rangle$. If $l = 1$, it simply indicates how many copies of $\langle i_1 \rangle$ are present in the system of systems.

The latter point deserves more explanation. Suppose we have a simple system of systems with only one path from $\langle 1 \rangle$ to $\langle 1, 1 \rangle$. Here, there are $n\langle 1 \rangle$ automata of type $\langle 1 \rangle$, but each contains $n\langle 1, 1 \rangle$ automata of type $\langle 1, 1 \rangle$. Therefore, there are in total $n\langle 1 \rangle + n\langle 1 \rangle \cdot n\langle 1, 1 \rangle$ automata in this system of systems. For example, the system of systems representation of a server farm consisting of a single farm made up of 10 servers, each hosting 20 jobs, will have 211 automata.

In order to describe the state of a system of systems, we will use a boolean vector of the form

$$\boldsymbol{b} := \left(b^j \langle \boldsymbol{i}_l \rangle [\boldsymbol{k}_l] \right), \forall \langle \boldsymbol{i}_l \rangle : \langle \boldsymbol{i}_l \rangle \in \mathcal{A}, 1 \leq j \leq d\langle \boldsymbol{i}_l \rangle, \tag{1}$$

where $\boldsymbol{k}_l = (k_1, \ldots, k_l)$ is such that $1 \leq k_m \leq n\langle i_1, \ldots, i_m \rangle$, for all $1 \leq m \leq l$. Each element $b^j \langle \boldsymbol{i}_l \rangle [\boldsymbol{k}_l]$ equals either 1 or 0. Specifically, $b^j \langle \boldsymbol{i}_l \rangle [\boldsymbol{k}_l] = 1$ if and only if j is the current local state of the automaton of type $\langle \boldsymbol{i}_l \rangle$ reached by taking the k_1-th copy of $\langle i_1 \rangle$, the k_2-th copy of $\langle i_1, i_2 \rangle$ and so on. Thus we record, for each copy of an automaton and for each local state of the automaton of type $\langle \boldsymbol{i}_l \rangle$, whether this instance is in that state or not. The double indexing $\langle \boldsymbol{i}_l \rangle$ and $[\boldsymbol{k}_l]$ of the vector $\boldsymbol{b}$ is required because we need to identify a specific automata of a specific automata class. Hence $\langle \boldsymbol{i}_l \rangle$ identifies the automata class in the system of systems tree, while $\boldsymbol{k}_l$ specifies the actual element of the population. In order to do this, we also need to know which are the actual automata containing a given agent, hence the need of another vector of coordinates.

Example 1 *We use the following running example to illustrate the definitions presented in this section. Let us consider the system of systems illustrated in Figure 1,with three automata classes,* $\langle 1 \rangle$, $\langle 1, 1 \rangle$, *and* $\langle 2 \rangle$, *with two local states each and with transitions*

$$\langle 1 \rangle^1 \to \langle 1 \rangle^2, \quad \langle 1 \rangle^2 \to \langle 1 \rangle^1,$$
$$\langle 2 \rangle^1 \to \langle 2 \rangle^2, \quad \langle 2 \rangle^2 \to \langle 2 \rangle^1,$$
$$\langle 1, 1 \rangle^1 \to \langle 1, 1 \rangle^2, \quad \langle 1, 1 \rangle^2 \to \langle 1, 1 \rangle^1.$$

Let us set $n\langle 1\rangle = 2$, $n\langle 2\rangle = 1$, *and* $n\langle 1,1\rangle = 2$*; that is, there are two copies of automaton* $\langle 1,1\rangle$ *within each of the two copies of automaton* $\langle 1\rangle$*. Therefore, the state descriptor has the form:*

$$\boldsymbol{b} = \Big(b^1\langle 1\rangle[1], b^2\langle 1\rangle[1], b^1\langle 1,1\rangle[1,1], b^2\langle 1,1\rangle[1,1], b^1\langle 1,1\rangle[1,2], b^2\langle 1,1\rangle[1,2],$$
$$b^1\langle 1\rangle[2], b^2\langle 1\rangle[2], b^1\langle 1,1\rangle[2,1], b^2\langle 1,1\rangle[2,1], b^1\langle 1,1\rangle[2,2], b^2\langle 1,1\rangle[2,2], b^1\langle 2\rangle[1], b^2\langle 2\rangle[1]\Big), \tag{2}$$

where the elements in the first line denote the local states of the automata reachable from the first copy of $\langle 1\rangle$*; all but the last two elements of the second line describe the local states of the automata reachable from the second copy of* $\langle 1\rangle$*; the last two elements denote the state of the only automaton* $\langle 2\rangle$*, which has no children.*

2.2 Semantics

The dynamics of a system of systems are impacted by two kind of interactions between its components: *horizontal interaction* and *vertical interaction*. Horizontal interaction comes about through the mutual influence of the dynamics of entities at the same level of the hierarchy. For instance, in a server farm model this can be dynamics of processes and jobs within a single computer. However, these dynamics are not independent of the context, but can be affected by the state of the containing node and by the state of the automata contained in the interacting ones. This is termed vertical interaction. Again, in the server farm example, the speed of processes in a computer may depend on its energy state, e.g. whether it is in power saving mode or not. Furthermore, two computers in a server farm can exchange jobs, if one has many of them waiting to be processed while the other is idle. Another form of vertical interaction is caused because a state change at one level in the tree can propagate its effects to its descendent nodes: think about the effect of a computer losing power will have on the processes running within it. We will describe the first kind of interaction by *events*, which are specified by rules at system level, while the second kind of dynamics will be described by a *causal map*.

The main source of dynamics of a system of systems are the events $\mathcal{E}$, which cause a change in the state descriptor (1). Each event $\eta \in \mathcal{E}$ defines horizontal interaction as a form of synchronisation between sibling automata in the system of systems tree. It is characterised by a *synchronisation set* S^η, specifying which class of automata and how many instances are involved in the event, and by a function F^η giving the rate of the interaction. Similarly to [12], this is a functional rate, in order to compactly describe the overall behaviour. We choose such rates to depend on the population levels only. This choice is based on the assumption that replicas of the same automaton are statistically equal to each other, but the state of each individual can be observed. However, the behaviour of an automaton can be dependent on the current state of its parent and of its siblings and children. Here such a relationship is expressed by the fact that functional rates may be dependent on the total populations of children and siblings. Formally, this may be achieved

by defining events $\mathcal{E}$ in the following form.

Definition 2.1 [Events] An event $\eta \in \mathcal{E}$ is a pair $\eta = (S^\eta, F^\eta)$, where:

- S^η is the synchronisation set containing automata transitions denoted by $\langle \boldsymbol{i}_l^s \rangle^j \to \langle \boldsymbol{i}_l^s \rangle^{j'}$, $s = 1, \ldots, s_\eta$, such that $\langle \boldsymbol{i}_{l-1}^{s_1} \rangle = \langle \boldsymbol{i}_{l-1}^{s_2} \rangle$ for each $s_1, s_2 = 1, \ldots, s_\eta$ (the automata involved in the event have the same parent), where s_η is the number of automata involved in the event. For simplicity, we also require $\langle \boldsymbol{i}_l^{s_1} \rangle \neq \langle \boldsymbol{i}_l^{s_2} \rangle$, i.e. the automata involved in the synchronisation are all distinct. [1]
- The function F^η gives the rate at which a specific tuple of automata in states $\langle \boldsymbol{i}_l^1 \rangle^j, \ldots, \langle \boldsymbol{i}_l^{s_\eta} \rangle^j$ performs the event. The function has parameters:

$$F^\eta\big(\boldsymbol{a}\langle \boldsymbol{i}_l^1 \rangle, \ldots, \boldsymbol{a}\langle \boldsymbol{i}_l^{s_\eta} \rangle, \boldsymbol{a}\langle \boldsymbol{i}_{l-1} \rangle, \boldsymbol{X}_s\langle \boldsymbol{i}_{l-1} \rangle, \boldsymbol{X}_c\langle \boldsymbol{i}_l^1 \rangle, \ldots, \boldsymbol{X}_c\langle \boldsymbol{i}_l^{s_\eta} \rangle\big),$$

where

- $\boldsymbol{a}\langle \boldsymbol{i}_l^s \rangle := \big(a^1\langle \boldsymbol{i}_l^s \rangle, \ldots, a^{d\langle \boldsymbol{i}_l^s \rangle}\langle \boldsymbol{i}_l^s \rangle\big)$ is the descriptor for the state space of $\langle \boldsymbol{i}_l^s \rangle$, for $s = 1, \ldots, s_\eta$;
- $\boldsymbol{a}\langle \boldsymbol{i}_{l-1} \rangle$ is the descriptor for the state space of the parent of $\langle \boldsymbol{i}_l^1 \rangle, \ldots, \langle \boldsymbol{i}_l^{s_\eta} \rangle$;
- $\boldsymbol{X}_s\langle \boldsymbol{i}_{l-1} \rangle$ is the state descriptor for the population of the siblings of $\langle \boldsymbol{i}_l^1 \rangle, \ldots, \langle \boldsymbol{i}_l^{s_\eta} \rangle$,

$$\boldsymbol{X}_s\langle \boldsymbol{i}_{l-1} \rangle := \left(\boldsymbol{X}\langle \boldsymbol{i}_{l-1}, 1 \rangle, \ldots, \boldsymbol{X}\langle \boldsymbol{i}_{l-1}, c\langle \boldsymbol{i}_{l-1} \rangle \rangle\right),$$

where

$$\boldsymbol{X}\langle \boldsymbol{m}_l \rangle := \left(X^1\langle \boldsymbol{m}_l \rangle, \ldots, X^{d\langle \boldsymbol{m}_l \rangle}\langle \boldsymbol{m}_l \rangle\right), \qquad \text{for all } \boldsymbol{m}_l : \langle \boldsymbol{m}_l \rangle \in \mathcal{A}$$

and $X^j\langle \boldsymbol{m}_l \rangle$ is the count of the number of instances of $\langle \boldsymbol{m}_l \rangle$ in state j;

- $\boldsymbol{X}_c\langle \boldsymbol{i}_l^s \rangle := (\boldsymbol{X}\langle \boldsymbol{i}_l^s, 1 \rangle, \ldots, \boldsymbol{X}\langle \boldsymbol{i}_l^s, c\langle \boldsymbol{i}_l^s \rangle \rangle)$ is the state descriptor for the population of children of $\langle \boldsymbol{i}_l^s \rangle$, for $s = 1, \ldots, s_\eta$.

Example 1 (continued) *Consider again the example illustrated in Figure 1 and described earlier. Let α denote an action at the top level, involving automata in state $\langle 1 \rangle^1$ and $\langle 2 \rangle^2$. We would write*

$$S^\alpha = \{\langle 1 \rangle^1 \to \langle 1 \rangle^2, \langle 2 \rangle^1 \to \langle 2 \rangle^2\}.$$

The associated function F^α has formal parameters

$$F^\alpha\Big(a^1\langle 1 \rangle, a^2\langle 1 \rangle, a^1\langle 2 \rangle, a^2\langle 2 \rangle, X^1\langle 1 \rangle, X^2\langle 1 \rangle, X^1\langle 2 \rangle, X^2\langle 2 \rangle, X^1\langle 1, 1 \rangle, X^2\langle 1, 1 \rangle\}\Big),$$

[1] This condition can be easily dropped, at the price of enforcing a minimum number of automata per class in the CTMC and making S^η a multi-set.

where the first line is related to the local states of two automata of kind $\langle 1\rangle$ *and* $\langle 2\rangle$*; the second line describes the dependence on the population of the sibling's local states, and the last line shows the dependence on the children's local states. Setting*

$$F^{\alpha} = \lambda_{\alpha} a^1\langle 1\rangle a^1\langle 2\rangle, \quad \textit{for } \lambda_{\alpha} > 0,$$

indicates that a synchronisation happens whenever the two automata are in their local state 1; when this occurs, the automata change their local state in 2, as expressed in the synchronisation set S^{α}*. The rate of the synchronisation is given by* λ_{α}*.*

In the above example, for simplicity we have considered a function, F^{α}, which does not depend on the populations of the parent automata or of the children automata. Section 4 will study a somewhat more elaborate case study where such dependencies are instead present. Furthermore, let us notice that we purposely use formal parameters denoted by the letter a (e.g., $a^1\langle 1\rangle$ and $a^1\langle 2\rangle$) to indicate that this is a *template* function that may be applied to distinct the elements of the boolean state descriptor vector $\boldsymbol{b}$ in (2). Although this will be defined more precisely later via Definiton 2.3, here we anticipate that this function will induce two distinct transitions for $\boldsymbol{b}$. More specifically, both will involve the same copy of automaton $\langle 2\rangle$; however, the first transition will be concerned with its interaction with the first copy of $\langle 1\rangle$, while the second transition will describe the interaction with the second copy of $\langle 2\rangle$.

An event specifies which automata are affected by a synchronisation, and how fast this happens. However, it says nothing about what actually happens their child nodes. Let us recall that describing this behaviour can be useful to model situations where an event in a system affects the behaviour of its inner components, e.g., a power outage for a computer will abort all of its software processes. Such a form of *vertical interaction* is instead captured by the notion of *causal map*, which defines how the transition of a parent automaton impacts on its child automata.

Definition 2.2 [Causal map] A causal map $\mathcal{C}$ is a set of rules of the form

$$\langle \boldsymbol{i}_l\rangle^j \to \langle \boldsymbol{i}_l\rangle^k \Rightarrow \langle \boldsymbol{i}_l, r_1\rangle^{j_1} \to \langle \boldsymbol{i}_l, r_1\rangle^{k_1} :: p_1; \ldots; \langle \boldsymbol{i}_l, r_m\rangle^{j_m} \to \langle \boldsymbol{i}_l, r_m\rangle^{k_m} :: p_m$$

where p_i is either a number in $(0, 1]$, or the special symbol !.

The values p_j attached to each automata event in the right hand side of a rule in the causal map specify either the probability with which the update happens (when $p \in (0, 1]$) or that exactly one child automaton changes state (if $p = !$).

A causal map is *well-formed* if each transition in the left hand side appears at most once in $\mathcal{C}$, and there are never two events out of the same state in the right hand side of a rule, i.e. $\langle \boldsymbol{i}_l, r_{s_1}\rangle^{j_{s_1}} \neq \langle \boldsymbol{i}_l, r_{s_2}\rangle^{j_{s_2}}$ for $s_1 \neq s_2$. In the following, we assume all causal maps are well-formed and we will denote by $rule(\langle \boldsymbol{i}_l\rangle^i \to \langle \boldsymbol{i}_l\rangle^j)$ the right hand side of the rule in $\mathcal{C}$, if any, having $\langle \boldsymbol{i}_l\rangle^i \to \langle \boldsymbol{i}_l\rangle^j$ as the left hand side.

Example 1 (continued) *For the previously considered example, we can define the*

casual map

$$\langle 1\rangle^1 \to \langle 1\rangle^2 \Rightarrow \langle 1,1\rangle^1 \to \langle 1,1\rangle^2 :: ! \tag{3}$$

This causal map says that, whenever the local state of one automaton $\langle 1\rangle$ *changes from 1 to 2, then* exactly one *child automaton* $\langle 1,1\rangle$ *will change from state 1 to state 2.*

As discussed, the specification of events is given at the level of automata classes. Recall, however, that a system of systems is comprised of a number of instances of each automaton class. Thus, we may think of a population corresponding to each class of automata. In practice, each event will involve some elements of the population of the given automata class. Hence we need a mechanism to identify the actual elements or agents involved. Furthermore, we need to take all possible choices of automata into account, each of which provides an *instance* of the event. More specifically, consider an event $\eta \in \mathcal{E}$, involving automata of classes $\langle \boldsymbol{i}_l^1\rangle, \ldots, \langle \boldsymbol{i}_l^{s_\eta}\rangle$. We define the event rate of a specific set of automata in the system of systems involved in η by an *instance function* $F^\eta[\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta}](\boldsymbol{b})$. As event η synchronises automata at level l, we need to identify the parent automaton in which it operates. This is indicated with $\langle \boldsymbol{i}_{l-1}^\eta\rangle$. The coordinates $\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta}$ specify the actual automata involved in the event. Each $\boldsymbol{k}_l^j = (k_1^j, \ldots, k_l^j)$ is such that $1 \le k_m^j \le n\langle \boldsymbol{i}_m^j\rangle$, with $1 \le m \le l$ and $j = 1 \ldots s_\eta$. Moreover, as the involved automata need to be contained in the same parent, it holds that $\boldsymbol{k}_{l-1}^i = \boldsymbol{k}_{l-1}^j$. A tuple $(\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta})$ that satisfies these constraints is called *an instance of the event* η, and represents one way in which the event may be manifest within the system of systems by choosing particular instances to take part in the event and undergo the updates. For any given event and given state of the system of systems there may be many different ways in which the event may be instantiated. The set of instances for η is denoted by K_η. Each function is associated with a *jump* or *update vector* that suitably changes the state descriptor, according to the synchronisation set S^η, i.e. each entry in $\boldsymbol{b}$ will be incremented or decremented by 1 to reflect the entry into and exit from local states within each instance of each automaton.

Example 1 (continued) *Recall that* α *is an action at the top level of the example shown in Figure 1, involving automata in state* $\langle 1\rangle^1$ *and* $\langle 2\rangle^2$*. The tuples*

$$\big(1,1\big) \qquad \textit{and} \qquad \big(2,1\big)$$

form the set of instances for the event α*. They indicate the possible interaction between either of the two copies of automaton* $\langle 1\rangle$ *and the only copy of automaton* $\langle 2\rangle$*.*

Before we define the dynamics of the CTMC associated with a system of systems, it will be useful to introduce some additional notation. The rich nature of the dynamics captured in systems of systems means that the rate, and indeed the existence of a transition, depend not only on the state of the automaton in which the transition occurs, but also potentially on the state of other automata at the same level (siblings), its enclosing automaton (parent) and the automata within it (children). But we do restrict that this dependence will only depend on siblings and

children through their population counts, based on the assumption that instances are indistinguishable.

We denote by $\boldsymbol{b}_{\langle \boldsymbol{i}_l^j \rangle}[\boldsymbol{k}_l^j]$ the vector of state entries of the form in equation (1), corresponding to instance $\boldsymbol{k}_l^j$:

$$\boldsymbol{b}_{\langle \boldsymbol{i}_l^j \rangle}[\boldsymbol{k}_l^j] := \left(b^1\langle \boldsymbol{i}_l^j \rangle[\boldsymbol{k}_l^j], \ldots, b^{d\langle \boldsymbol{i}_l^j \rangle}\langle \boldsymbol{i}_l^j \rangle[\boldsymbol{k}_l^j] \right), \quad j = 1 \ldots s_\eta. \tag{4}$$

As discussed above, the transition in the CTMC will depend on this detailed current state of the instances involved in the event, but may also be affected by the sibling and child automata through their aggregate state, in terms of their populations. Thus we introduce notation to represent these populations, where $\boldsymbol{S}_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}]$ is the vector of counts corresponding to all siblings. Essentially, for each automata class contained in the instance $\boldsymbol{k}_{l-1}$ of the parent automata class $\langle \boldsymbol{i}_{l-1} \rangle$, we count how many automata instances are in each local state. $\boldsymbol{S}^j_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}]$, in particular, is the vector of counts across states of the automata class $\langle \boldsymbol{i}_{l-1}, j \rangle$.

$$\boldsymbol{S}_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}] := \left(\boldsymbol{S}^1_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}], \ldots, \boldsymbol{S}^{c\langle \boldsymbol{i}_{l-1} \rangle}_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}] \right), \tag{5}$$

$$\boldsymbol{S}^j_{\langle \boldsymbol{i}_{l-1} \rangle}[\boldsymbol{k}_{l-1}] := \left(\sum_{r=1}^{n\langle \boldsymbol{i}_{l-1}, j \rangle} b^1_{\langle \boldsymbol{i}_{l-1}, j \rangle}[\boldsymbol{k}_{l-1}, r], \ldots, \sum_{r=1}^{n\langle \boldsymbol{i}_{l-1}, j \rangle} b^{d\langle \boldsymbol{i}_{l-1}, j \rangle}_{\langle \boldsymbol{i}_{l-1}, j \rangle}[\boldsymbol{k}_{l-1}, r] \right), \tag{6}$$

Similarly we take into consideration the populations within the child automata, $\boldsymbol{C}_{\langle \boldsymbol{i}_l^j \rangle}, [\boldsymbol{k}_l^j]$, which counts the states of automata contained in the automata $\boldsymbol{k}_l^j$:

$$\boldsymbol{C}_{\langle \boldsymbol{i}_l^j \rangle}[\boldsymbol{k}_l^j] := \boldsymbol{S}_{\langle \boldsymbol{i}_l^j \rangle}[\boldsymbol{k}_l^j] \tag{7}$$

Example 1 (continued) *For event α, we have that*

$$\begin{aligned}
\boldsymbol{b}_{\langle 1 \rangle}[1] &= \left(b^1\langle 1 \rangle[1], b^2\langle 1 \rangle[1] \right), \\
\boldsymbol{b}_{\langle 1 \rangle}[2] &= \left(b^1\langle 1 \rangle[2], b^2\langle 1 \rangle[2] \right), \\
\boldsymbol{b}_{\langle 2 \rangle}[1] &= \left(b^1\langle 2 \rangle[1], b^2\langle 2 \rangle[1] \right).
\end{aligned}$$

The vector of the population of siblings is given by

$$\left(b^1\langle 1 \rangle[1] + b^1\langle 1 \rangle[2], b^2\langle 1 \rangle[1] + b^2\langle 1 \rangle[2], b^1\langle 2 \rangle[1], b^2\langle 2 \rangle[1] \right).$$

Similarly, the vector of the populations of children is

$$\left(b^1\langle 1, 1 \rangle[1, 1] + b^1\langle 1, 1 \rangle[1, 2], b^2\langle 1, 1 \rangle[1, 1] + b^2\langle 1, 1 \rangle[1, 2] \right).$$

We are now ready to provide the definition of the actual dynamics of the CTMC. What we need to do is to specify, for each instance of each event η, the rate (given by the instance function, where we substitute the correct population counts of siblings

and child nodes) and the update vector, specifying the net change in the CTMC state descriptor. Furthermore, each event propagates its effects downstream in the automata hierarchy, according to the causal rules associated with a state change in the automata involved in the transition. Hence, we also need to specify at what rate child automata perceive the event that happened at their parent level, and how their state is modified.

Definition 2.3 [CTMC Dynamics] Let $\mathcal{A}$ be a system of systems with events $\mathcal{E}$. Let (1) define the CTMC state descriptor. Then, the transition functions for the CTMC are induced from the events $\eta \in \mathcal{E}$ as follows:

$$F^{\eta}[\boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}](\boldsymbol{b}) := F^{\eta}\big(\boldsymbol{b}_{\langle \boldsymbol{i}_l^1\rangle}[\boldsymbol{k}_l^1],\dots,\boldsymbol{b}_{\langle \boldsymbol{i}_l^{s_\eta}\rangle}[\boldsymbol{k}_l^{s_\eta}],\boldsymbol{b}_{\langle \boldsymbol{i}_{l-1}\rangle}[\boldsymbol{k}_{l-1}], \boldsymbol{S}_{\langle \boldsymbol{i}_{l-1}\rangle}[\boldsymbol{k}_{l-1}],\boldsymbol{C}_{\langle \boldsymbol{i}_l^1\rangle}[\boldsymbol{k}_l^1],\dots,\boldsymbol{C}_{\langle \boldsymbol{i}_l^{s_\eta}\rangle}[\boldsymbol{k}_l^{s_\eta}]\big), \quad (8)$$

for all $(\boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}) \in K_\eta$, with $\boldsymbol{k}_l^j = (k_1^j,\dots,k_l^j)$ such that $1 \le h_m^j \le n\langle \boldsymbol{i}_m^j\rangle$, with $1 \le m \le l$ and $j = 1 \dots s_\eta$.

The associated jump vector, denoted by $\boldsymbol{e}^{\eta}[\boldsymbol{k_l^1},\dots,\boldsymbol{k_l^{s_\eta}}] := \Big(e^j\langle \boldsymbol{i}_l'\rangle[\boldsymbol{h}_l]\Big), \forall \langle \boldsymbol{i}_l'\rangle$ such that $\langle \boldsymbol{i}_l'\rangle \in \mathcal{A}$, with $1 \le j \le d\langle \boldsymbol{i}_l'\rangle, 1 \le h \le n\langle \boldsymbol{i}_l'\rangle$, is defined as follows:

$$e^j\langle \boldsymbol{i}_l'\rangle[\boldsymbol{h}_l] = \begin{cases} +1 & \text{if } \langle \boldsymbol{i}_l^r\rangle^{r_1} \to \langle \boldsymbol{i}_l^r\rangle^{r_2} \in S^\eta, \boldsymbol{i}_l' = \boldsymbol{i}_l^r, \boldsymbol{h}_l = \boldsymbol{k}_l^r, j = r_2, \\ -1 & \text{if } \langle \boldsymbol{i}_l^r\rangle^{r_1} \to \langle \boldsymbol{i}_l^r\rangle^{r_2} \in S^\eta, \boldsymbol{i}_l' = \boldsymbol{i}_l^r, \boldsymbol{h}_l = \boldsymbol{k}_l^r, j = r_1 \\ 0 & \text{otherwise.} \end{cases}$$

Then, for each $\langle \boldsymbol{i}_l^r\rangle^{r_1} \to \langle \boldsymbol{i}_l^r\rangle^{r_2} \in S^\eta$, such that there is a rule for it in $\mathcal{C}$, say $R = rule(\langle \boldsymbol{i}_l^r\rangle^{r_1} \to \langle \boldsymbol{i}_l^r\rangle^{r_2})$, we define a transition function for each copy of the automaton child m of type $\langle \boldsymbol{i}_l^r, r'\rangle$, contained in the automaton reachable by $\langle \boldsymbol{k}_l\rangle$, for all $1 \le m \le n\langle \boldsymbol{i}_l^r, r'\rangle$, as follows:

$$F_{+1}^{\eta}\langle \boldsymbol{i}_l^r, s\rangle[\boldsymbol{k}_l^r, m \mid \boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}](\boldsymbol{b}) := \begin{cases} \dfrac{F^{\eta}[\boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}](\boldsymbol{b}) \cdot b^d_{\langle \boldsymbol{i}_l^r,s\rangle}[\boldsymbol{k}_l^r, m]}{\sum_{t=1}^{n\langle \boldsymbol{i}_l^r,s\rangle} b^d_{\langle \boldsymbol{i}_l^r,s\rangle}[\boldsymbol{k}_l^r, t]} & \text{if } \langle \boldsymbol{i}_l^r, s\rangle^d \to \langle \boldsymbol{i}_l^r, s\rangle^f :: ! \in R, \\ F^{\eta}[\boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}] \cdot p \cdot b^d_{\langle \boldsymbol{i}_l^r,s\rangle}[\boldsymbol{k}_l^r, m] & \text{if } \langle \boldsymbol{i}_l^r, s\rangle^d \to \langle \boldsymbol{i}_l^r, s\rangle^f :: p \in R, \\ 0 & \text{otherwise,} \end{cases} \quad (9)$$

and the corresponding jump vector

$$\boldsymbol{e}_{+1}^{\eta}\langle \boldsymbol{i}_l^r, s\rangle[\boldsymbol{k_l}, m \mid \boldsymbol{k}_l^1,\dots,\boldsymbol{k}_l^{s_\eta}] := \Big(e^j\langle \boldsymbol{i}_l'\rangle[\boldsymbol{h}_l]\Big),$$

with components

$$e^j\langle \boldsymbol{i}'_l\rangle[\boldsymbol{h}_l] = \begin{cases} +1 & \text{if } \langle \boldsymbol{i}^r_l, s\rangle^d \to \langle \boldsymbol{i}^r_l, s\rangle^f \in R, j = d, \boldsymbol{i}'_l = \langle \boldsymbol{i}^r_l, s\rangle, \boldsymbol{h}_l = \langle \boldsymbol{k}^r_l, m\rangle \\ -1 & \text{if } \langle \boldsymbol{i}^r_l, s\rangle^d \to \langle \boldsymbol{i}^r_l, s\rangle^f \in R, j = f, \boldsymbol{i}'_l = \langle \boldsymbol{i}^r_l, s\rangle, \boldsymbol{h}_l = \langle \boldsymbol{k}^r_l, m\rangle \\ 0 & \text{otherwise.} \end{cases}$$

Equation (8) represents an *instance* of the transition function to describe the behaviour of a specific tuple of automata participating in the event. It depends on the local state of the automata involved, cf. (4), and on the state of its parent, which is reached by removing the last element of the position vector $\boldsymbol{k}_l$; the rate may also depend on the automaton's siblings, cf. (5). Note that, by definition, the siblings are those that can be found within the automaton's parent, i.e., within the specific copy where the automaton lives. As discussed, the dependence on the siblings' state is through the total population of automata in a given local state, cf. (6). Finally, the rate may be dependent on the automata's children in a similar manner. Equation (9) considers the impact on the children. If only one child in a given state is moved, then the rate is adjusted by dividing it by the total number of children. Hence, we are assuming that one child is selected uniformly at random to be updated (this is justified by the assumption of indistinguishability). If each child is selected to move with probability p, then each child sees a fraction p of the total rate.

Example 1 (continued) *Let us apply the above definitions to our running example. It holds that*

$$F^\alpha[1,1](\boldsymbol{b}) = \lambda_\alpha b^1\langle 1\rangle[1]b^1\langle 2\rangle[1],$$
$$F^\alpha[2,1](\boldsymbol{b}) = \lambda_\alpha b^1\langle 1\rangle[2]b^1\langle 2\rangle[1],$$

reflecting the possibility of the only copy of automaton $\langle 2\rangle$ to interact with either of the two copies of automaton $\langle 1\rangle$. Using the same ordering as in (2) for the state descriptor, these two functions give rise to jump vectors equal to

$$\big(-1,+1,0,0,0,0,0,0,0,0,0,0,-1,+1\big)$$

and

$$\big(0,0,0,0,0,0,-1,+1,0,0,0,0,-1,+1\big),$$

respectively.

Because of the casual map (3), α has an impact on the child automata $\langle 1,1\rangle$. For instance, the functions

$$F^\alpha_{+1}\langle 1,1\rangle[1,1 \mid 1,1](\boldsymbol{b}) = \lambda_\alpha b^1\langle 1\rangle[1]b^1\langle 2\rangle[1]\frac{b^1\langle 1,1\rangle[1,1]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,1\rangle[1,2]}$$

and

$$F^\alpha_{+1}\langle 1,1\rangle[1,2 \mid 1,1](\boldsymbol{b}) = \lambda_\alpha b^1\langle 1\rangle[1]b^1\langle 2\rangle[1]\frac{b^1\langle 1,1\rangle[1,2]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,2\rangle[1,2]}$$

give the rate at which each copy of $\langle 1,1\rangle$ *in the first copy of automaton* $\langle 1\rangle$ *sees an* α*-event. Analogous functions are defined for the second copy of* $\langle 1\rangle$*. The respective jump vectors are then*

$$\big(0,0,-1,+1,0,0,0,0,0,0,0,0,0,0\big)$$

and

$$\big(0,0,0,0,-1,+1,0,0,0,0,0,0,0,0\big).$$

3 Fluid Equations

We now construct a set of differential equations providing a first order approximation of the average evolution of the CTMC. We will define this set of ODEs approximating the expectation of the state variables $\boldsymbol{b} = \Big(b^j\langle \boldsymbol{i}_l\rangle[\boldsymbol{k}_l]\Big)$. These variables, in fact, determine the population variables according to (6). As the state variables $\boldsymbol{b}$ take values in $\{0,1\}$, approximating the expectation corresponds to approximating the probability of a (random) automaton being in a given state. This is similar to the spatial mean field for Markovian agents considered in [6], although here we provide a different derivation of the fluid approximation.

More specifically, the set of fluid ODEs is constructed, as customary [3], using the drift vector, which describes the instantaneous average variation of system variables in a given state. For an arbitrary system of systems, the drift is defined by

$$F(\boldsymbol{b}) := \sum_{\eta\in\mathcal{E}} \sum_{(\boldsymbol{k}_l^1,\ldots,\boldsymbol{k}_l^{s_\eta})\in K_\eta} \Bigg(\boldsymbol{e}^{\,\eta}[\boldsymbol{k}_{\boldsymbol{l}}^{\boldsymbol{1}},\ldots,\boldsymbol{k}_{\boldsymbol{l}}^{\boldsymbol{s_\eta}}]F^\eta[\boldsymbol{k}_l^1,\ldots,\boldsymbol{k}_l^{s_\eta}](\boldsymbol{b}) + \sum_{r=1}^{s_\eta}\sum_{c=1}^{c\langle \boldsymbol{i}_l^r\rangle}\sum_{m=1}^{n\langle \boldsymbol{i}_l^r,c\rangle} \boldsymbol{e}^\eta_{+1}\langle \boldsymbol{i}_l^r,c\rangle[\boldsymbol{k_l},m \mid \boldsymbol{k}_l^1,\ldots,\boldsymbol{k}_l^{s_\eta}]F^\eta_{+1}\langle \boldsymbol{i}_l^r,c\rangle[\boldsymbol{k}_l^r,m \mid \boldsymbol{k}_l^1,\ldots,\boldsymbol{k}_l^{s_\eta}](\boldsymbol{b})\Bigg) \tag{10}$$

The summations in the drift equation take into account, for each transition η, all its possible instances at the level of interacting automata and all its possible impacts on their children.

Then, the fluid ODE equation is simply

$$\frac{d\boldsymbol{b}(t)}{dt} = F(\boldsymbol{b}(t)). \tag{11}$$

This equation can be seen as an approximate equation for the average of the CTMC. Indeed, the true equation for the average, as obtained from the Dynkin formula (see, for instance, [13]), is

$$\frac{d\mathbb{E}[\boldsymbol{b}(t)]}{dt} = \mathbb{E}[F(\boldsymbol{b}(t))].$$

Hence the fluid equation can be obtained by "pushing" the expectation inside the (generally non-linear) function F. This operation corresponds to a first-order approximation of the real equation [2,7]. Approximate fluid estimates of performability

measures can be expressed as appropriate deterministic functions (i.e., *rewards*) of the solutions of (11), as discussed, for example, in [14].

We stress here that this simple definition of the drift and of the fluid equation is possible because of the notation carefully introduced in the previous section.

Example 1 (continued) *For the running example, let us assume that α is the only event defined. Then the system of ODEs corresponding to the example, expressed in components, is:*[2]

$$
\begin{aligned}
\dot{b}^1\langle 1\rangle[1] &= -\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \\
\dot{b}^2\langle 1\rangle[1] &= +\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \\
\dot{b}^1\langle 1,1\rangle[1,1] &= -\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[1,1]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,1\rangle[1,2]} \\
\dot{b}^2\langle 1,1\rangle[1,1] &= +\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[1,1]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,1\rangle[1,2]} \\
\dot{b}^1\langle 1,1\rangle[1,2] &= -\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[1,2]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,1\rangle[1,2]} \\
\dot{b}^2\langle 1,1\rangle[1,2] &= +\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[1,2]}{b^1\langle 1,1\rangle[1,1] + b^1\langle 1,1\rangle[1,2]} \\
\dot{b}^1\langle 1\rangle[2] &= -\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \\
\dot{b}^2\langle 1\rangle[2] &= +\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \\
\dot{b}^1\langle 1,1\rangle[2,1] &= -\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[2,1]}{b^1\langle 1,1\rangle[2,1] + b^1\langle 1,1\rangle[2,2]} \\
\dot{b}^2\langle 1,1\rangle[2,1] &= +\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[2,1]}{b^1\langle 1,1\rangle[2,1] + b^1\langle 1,1\rangle[2,2]} \\
\dot{b}^1\langle 1,1\rangle[2,2] &= -\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[2,2]}{b^1\langle 1,1\rangle[2,1] + b^1\langle 1,1\rangle[2,2]} \\
\dot{b}^2\langle 1,1\rangle[2,2] &= +\lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \frac{b^1\langle 1,1\rangle[2,2]}{b^1\langle 1,1\rangle[2,1] + b^1\langle 1,1\rangle[2,2]} \\
\dot{b}^1\langle 2\rangle[1] &= -\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] - \lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1] \\
\dot{b}^2\langle 2\rangle[1] &= +\lambda_\alpha b^1\langle 1\rangle[1] b^1\langle 2\rangle[1] + \lambda_\alpha b^1\langle 1\rangle[2] b^1\langle 2\rangle[1]
\end{aligned}
$$

To show how to obtain a differential model with size (i.e. number of equations) independent of the number of instances of automata, let us compare, for instance, the equations $\dot{b}^1\langle 1\rangle[1]$ and $\dot{b}^1\langle 1\rangle[2]$. Assuming that the initial conditions are such that $b^1\langle 1\rangle[1](0) = b^1\langle 1\rangle[1](0)$, it holds that the derivatives are equal at all future time points. By uniqueness of the solution, this implies that the two solutions are also equal. Thus it follows that one of the two equations can be removed. Moreover, this can be done systematically, by inspecting the definition of the drift and of the instance functions F^η and F^η_{+1}. We can easily see that, syntactically, the equations are the same for each tuple $(\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta}) \in K_\eta$. This readily implies

[2] Where, for compactness, we use the notation $\dot{b}$ for derivative.

that the equation for the drift is invariant under any permutation of agents that is consistent with the class structure. In particular, the fluid equations (i.e. the derived ODEs) for two agents of the same class are syntactically the same. This observation can be readily turned into the following.

Theorem 3.1 *Assume that for all* $\langle \boldsymbol{i}_l \rangle$, $\boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}_l](0) = \boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}'_l](0)$ *for any two* $\boldsymbol{k}'_l$ *and* $\boldsymbol{k}_l$*, and that the solution of (11) exists and is unique in* $[0, T]$.
Then, it holds that, for all $t \in [0, T]$

$$\boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}_l](t) = \boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k'}_l](t).$$

Proof. [Sketch] By invariance under permutation of agents of the same class, it follows that if $\boldsymbol{b}$ is *class invariant*, i.e. $\boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}_l] = \boldsymbol{b}\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}'_l]$, for any $\langle \boldsymbol{i}_l \rangle$, $\boldsymbol{k}_l$, $\boldsymbol{k}'_l$, then

$$F_{\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}_l]}(\boldsymbol{b}) = F_{\langle \boldsymbol{i}_l \rangle[\boldsymbol{k}'_l]}(\boldsymbol{b}),$$

i.e. the vector field is also class invariant. It follows that, if $\boldsymbol{b}$ is class invariant, then so is $\boldsymbol{b}+hF(\boldsymbol{b})$. Fix $h > 0$ and let $\boldsymbol{x}(0) = \boldsymbol{b}(0)$, and $\boldsymbol{x}((k+1)h) = \boldsymbol{x}(kh)+F(\boldsymbol{x}(kh))$. By an easy induction, we can establish that $\boldsymbol{x}(kh)$ is class invariant for any k. It follows that class invariance is preserved by the Euler integration scheme. Hence, by letting $h \to 0$, it is also preserved by ODE solutions. □ □

The simplified equations can be constructed directly by modifying the template expressions, observing that $S^j_{\langle \boldsymbol{i}_l \rangle}[\boldsymbol{h}_{l-1}] = S^j\langle \boldsymbol{i}_{l-1} \rangle = n\langle \boldsymbol{i}_l \rangle \hat{b}^j\langle \boldsymbol{i}_l \rangle$:

$$F^\eta\langle \boldsymbol{i}_l^j \rangle(\hat{\boldsymbol{b}}) := \prod_{r \neq j, r=1}^{s_\eta} n\langle \boldsymbol{i}_l^r \rangle F^\eta\Big(\hat{\boldsymbol{b}}\langle \boldsymbol{i}_l^1 \rangle, \ldots, \hat{\boldsymbol{b}}\langle \boldsymbol{i}_l^{k_\eta} \rangle, \hat{\boldsymbol{b}}\langle \boldsymbol{i}_{l-1} \rangle, \boldsymbol{S}\langle \boldsymbol{i}_{l-1} \rangle, \boldsymbol{S}\langle \boldsymbol{i}_l^1 \rangle, \ldots, \boldsymbol{S}\langle \boldsymbol{i}_l^{s_\eta} \rangle\Big). \tag{12}$$

This new template equation is obtained summing over all possible siblings of an agent of class $\langle \boldsymbol{i}_l^j \rangle$, using the assumption that agents involved in a synchronisation all belong to different classes. The multiplicative factor $\left(\prod_{r \neq j, r=1}^{s_\eta} n\langle \boldsymbol{i}_l^r \rangle\right)$ is the consequence of the fact that $F^\eta[\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta}](\boldsymbol{b}(t)) = F^\eta[\tilde{\boldsymbol{k}}_l^1, \ldots, \tilde{\boldsymbol{k}}_l^{s_\eta}](\boldsymbol{b}(t))$ for all $(\boldsymbol{k}_l^1, \ldots, \boldsymbol{k}_l^{s_\eta}) \neq (\tilde{\boldsymbol{k}}_l^1, \ldots, \tilde{\boldsymbol{k}}_l^{s_\eta})$ in K_η, and for each fixed agent of class $\langle \boldsymbol{i}_l^j \rangle$, there are $\left(\prod_{r \neq j, r=1}^{s_\eta} n\langle \boldsymbol{i}_l^r \rangle\right)$ instances of η.

4 Numerical Validation

In this section we present some numerical validation of the fluid approximation scheme presented in the previous section.

4.1 Model Description

To illustrate our framework, let us consider a performability model of a system of systems with four automata classes arranged as follows. There are two top-level two-state automata, $\langle 1 \rangle$ and $\langle 2 \rangle$, which represent the model of a computer and a

Symbol	*Meaning*	*Synchronisation set*	*Function*
α_1	Computer/user	$S^{\alpha_1} = \{\langle 1\rangle^1 \to \langle 1\rangle^1, \langle 2\rangle^1 \to \langle 2\rangle^2\}$	$F^{\alpha_1} = \lambda_{\alpha_1} a^1\langle 1\rangle a^1\langle 2\rangle$
α_2	User delay	$S^{\alpha_2} = \{\langle 2\rangle^2 \to \langle 2\rangle^1\}$	$F^{\alpha_2} = \lambda_{\alpha_2} a^2\langle 2\rangle$
α_3	Thread/CPU	$S^{\alpha_3} = \{\langle 1,1\rangle^2 \to \langle 1,1\rangle^1$	$F^{\alpha_3} = \lambda_{\alpha_3} \frac{a^2\langle 1,1\rangle}{X^2\langle 1,1\rangle} \frac{a^1\langle 1,2\rangle}{X^1\langle 1,2\rangle} \mathcal{X}$
		$\langle 1,2\rangle^1 \to \langle 1,2\rangle^1\}$	(where $\mathcal{X} = \min(X^2\langle 1,1\rangle, X^1\langle 1,2\rangle)$)
$\varphi_{\langle 1\rangle}$	Computer failure	$S^{\varphi_{\langle 1\rangle}} = \{\langle 1\rangle^1 \to \langle 1\rangle^2\}$	$F^{\varphi_{\langle 1\rangle}} = \lambda_{\varphi_{\langle 1\rangle}} a^1\langle 1\rangle$
$\rho_{\langle 1\rangle}$	Computer repair	$S^{\rho_{\langle 1\rangle}} = \{\langle 1\rangle^2 \to \langle 1\rangle^1\}$	$F^{\rho_{\langle 1\rangle}} = \lambda_{\rho_{\langle 1\rangle}} a^2\langle 1\rangle$
$\varphi_{\langle 1,1\rangle}$	Thread failure	$S^{\varphi_{\langle 1,1\rangle}} = \{\langle 1,1\rangle^1 \to \langle 1,1\rangle^3\}$	$F^{\varphi_{\langle 1,1\rangle}} = \lambda_{\varphi_{\langle 1,1\rangle}} a^1\langle 1,1\rangle$
$\rho_{\langle 1,1\rangle}$	Thread repair	$S^{\rho_{\langle 1,1\rangle}} = \{\langle 1,1\rangle^3 \to \langle 1,1\rangle^1\}$	$F^{\rho_{\langle 1,1\rangle}} = \lambda_{\rho_{\langle 1,1\rangle}} a^3\langle 1,1\rangle$
$\varphi_{\langle 1,2\rangle}$	CPU failure	$S^{\varphi_{\langle 1,2\rangle}} = \{\langle 1,2\rangle^1 \to \langle 1,2\rangle^2\}$	$F^{\varphi_{\langle 1,2\rangle}} = \lambda_{\varphi_{\langle 1,2\rangle}} a^1\langle 1,2\rangle$
$\rho_{\langle 1,2\rangle}$	CPU repair	$S^{\rho_{\langle 1,2\rangle}} = \{\langle 1,2\rangle^2 \to \langle 1,2\rangle^1\}$	$F^{\rho_{\langle 1,2\rangle}} = \lambda_{\rho_{\langle 1,2\rangle}} a^2\langle 1,2\rangle$

Table 1
Model equations. The parameters λ_β, for all symbols β are given positive reals.

user, respectively. Each computer has two automata children $\langle 1,1\rangle$ and $\langle 1,2\rangle$, which model software threads and CPUs respectively. Users interact with computers by issuing requests, interposing some think time between successive requests. Whenever a request arrives, a thread is acquired which is triggered to execute on a CPU. When execution is finished, the thread becomes idle again and ready to serve another request. Computers, threads, and CPUs may be subject to failure, which is intended to be a logical fault that can be recovered after some time. Formally, we describe the overall model with the automata given by

$$\begin{array}{lll}
\langle 1\rangle^1 \to \langle 1\rangle^1, & \langle 1\rangle^1 \to \langle 1\rangle^2, & \langle 1\rangle^2 \to \langle 1\rangle^1, \\
\langle 2\rangle^1 \to \langle 2\rangle^1, & \langle 2\rangle^2 \to \langle 1\rangle^1, & \\
\langle 1,1\rangle^1 \to \langle 1,1\rangle^2, & \langle 1,1\rangle^1 \to \langle 1,1\rangle^3, & \\
\langle 1,1\rangle^2 \to \langle 1,1\rangle^1, & \langle 1,1\rangle^3 \to \langle 1,1\rangle^1 & \\
\langle 1,2\rangle^1 \to \langle 1,2\rangle^1, & \langle 1,2\rangle^1 \to \langle 1,2\rangle^2, & \langle 1,2\rangle^2 \to \langle 1,2\rangle^1.
\end{array}$$

The events are specified in Table 1. Computer/user interaction is defined according to the law of mass action, which has been used to model connectivity by means of wireless networks [16]. The states labelled 1 in the automata $\langle 1\rangle$ and $\langle 2\rangle$ are therefore assumed to be the operational states where the interaction is possible. The causal map

$$\langle 1\rangle^1 \to \langle 1\rangle^1 \Rightarrow \langle 1,1\rangle^1 \to \langle 1,1\rangle^2 :: !$$

models that, upon a computer/user interaction, one thread that is initially idle (state $\langle 1,1\rangle^1$) starts processing on one CPU, by moving to state $\langle 1,1\rangle^2$. After a user issues a request it moves into state $\langle 2\rangle^2$, where it stays with an average

$b^1\langle 1\rangle$			$b^1\langle 1,1\rangle$			$b^2\langle 1,1\rangle$			$b^1\langle 1,2\rangle$			$b^1\langle 2\rangle$		
Min	*Avg.*	*Max*	*Min*	*Avg.*	*Max*	*Min*	*Avg.*	*Max*	*Min*	*Avg.*	*Max*	*Min*	*Avg.*	*Max*
< 0.01	0.14	3.18	< 0.01	0.45	13.40	< 0.01	0.16	7.57	< 0.01	0.02	0.27	< 0.01	0.62	20.61

Table 2
Relative density errors between ODE solutions and simulation over 500 randomly generated models.

delay $1/\lambda_{\alpha_2}$. That is, we are modelling a closed workload of $n\langle 2\rangle$ users interposing independent delays. Thread/CPU interaction, modelled by event α_3, is related by a dynamics consistent with a multi-server exponential queue with $X^2\langle 1,1\rangle$ jobs and $X^1\langle 1,2\rangle$ servers with individual service rate equal to λ_{α_3}. The fractions $a^2\langle 1,1\rangle/X^2\langle 1,1\rangle$ and $a^1\langle 1,2\rangle/X^1\langle 1,2\rangle$ indicate the probability that a specific pair of job/server $a^2\langle 1,1\rangle/a^1\langle 1,2\rangle$ are involved. Failure and repair events are assumed to be independent. Notice that, for simplicity, threads are assumed not to fail while they are executing on the CPU (state $\langle 1,1\rangle^2$), but only when they are idle (state $\langle 1,1\rangle^1$).

The aggregated ODE system, exploiting symmetries and multiplicities as in Equation (12), is

$$\begin{aligned}
\dot{b}^1\langle 1\rangle &= -\lambda_{\alpha_1} b^1\langle 1\rangle b^1\langle 2\rangle - \lambda_{\varphi_{\langle 1\rangle}} b^1\langle 1\rangle + \lambda_{\rho_{\langle 1\rangle}} b^2\langle 1\rangle \\
\dot{b}^1\langle 2\rangle &= -\lambda_{\alpha_1} b^1\langle 1\rangle b^1\langle 2\rangle + \lambda_{\alpha_2} b^2\langle 2\rangle \\
\dot{b}^1\langle 1,1\rangle &= -\frac{\lambda_{\alpha_1}}{n\langle 1,1\rangle} b^1\langle 1\rangle b^1\langle 2\rangle - \lambda_{\varphi_{\langle 1,1\rangle}} b^1\langle 1,1\rangle \\
&\quad + \frac{\lambda_{\alpha_3}}{n\langle 1,1\rangle} \min(b^2\langle 1,1\rangle n\langle 1,1\rangle, b^1\langle 1,2\rangle n\langle 1,2\rangle) + \lambda_{\rho_{\langle 1,1\rangle}} b^3\langle 1,1\rangle \\
\dot{b}^2\langle 1,1\rangle &= -\frac{\lambda_{\alpha_3}}{n\langle 1,1\rangle} \min(b^2\langle 1,1\rangle n\langle 1,1\rangle, b^1\langle 1,2\rangle n\langle 1,2\rangle) + \frac{\lambda_{\alpha_1}}{n\langle 1,1\rangle} b^1\langle 1\rangle b^1\langle 2\rangle \\
\dot{b}^1\langle 1,2\rangle &= -\lambda_{\varphi_{\langle 1,2\rangle}} b^1\langle 1,2\rangle + \lambda_{\rho_{\langle 1,2\rangle}} b^2\langle 1,2\rangle
\end{aligned}$$

with

$$\begin{aligned}
\dot{b}^2\langle 1\rangle &= -\dot{b}^1\langle 1\rangle \\
\dot{b}^2\langle 2\rangle &= -\dot{b}^1\langle 2\rangle \\
\dot{b}^3\langle 1,1\rangle &= -\dot{b}^1\langle 1,1\rangle - \dot{b}^2\langle 1,1\rangle \\
\dot{b}^2\langle 1,2\rangle &= -\dot{b}^1\langle 1,2\rangle.
\end{aligned}$$

These last equations hold because of conservation of mass.

4.2 Numerical Experiments

To assess the quality of the approximation, a numerical investigation was conducted on 500 model instances with randomly generated population sizes and rate parameters. Specifically, $n\langle 1\rangle$, $n\langle 2\rangle$, $n\langle 1,1\rangle$ were chosen randomly in the range $1,\ldots,10$, whereas $n\langle 1,2\rangle$ was chosen in the range $10,\ldots,50$, in order to model situations

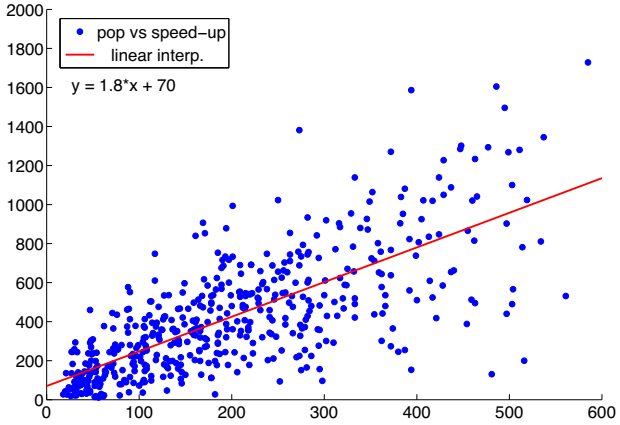

Fig. 2. Scatter plot of total population versus speed-up for the validation dataset. The red line is the linear regression.

where there are on average more threads than processors in a computer. The parameter rates were drawn uniformly at random as follows:

$$\begin{aligned} \lambda_{\alpha_1}, \lambda_{\alpha_3} &\in [0.01, 1], \\ \lambda_{\alpha_2}, \lambda_{\varphi_{\langle 1 \rangle}}, \lambda_{\varphi_{\langle 1,1 \rangle}}, \lambda_{\varphi_{\langle 1,2 \rangle}} &\in [0.001, 0.1], \\ \lambda_{\rho_{\langle 1 \rangle}}, \lambda_{\rho_{\langle 1,1 \rangle}}, \lambda_{\rho_{\langle 1,2 \rangle}} &\in [0.01, 0.50]. \end{aligned}$$

Such a design of the parameter space ensured that the model was exercised under a variety of operating regimes, e.g., different workloads of users and different levels of utilisation of the CPUs and threads.

In these tests we focussed on the model's steady-state behaviour. We analysed the approximation error by comparing the estimates obtained by stochastic simulation of the CTMC (the large state spaces prevented us from performing numerical solution) against ODE numerical integration. Stochastic simulation was conducted with the method of batch means with 5% confidence level at 95% confidence interval. The ODEs were solved using the well-known Runge-Kutta scheme as implemented in Matlab 7.9.0 through the function `ode45`. As an estimate of equilibrium, each ODE system was solved until a time point T at which the Euclidean norm of the derivative $\|d\boldsymbol{b}(T)/dt\|$ was less than 1E-8. Let $O^j\langle \boldsymbol{i} \rangle$ (resp., $S^j\langle \boldsymbol{i} \rangle$) be the ODE (resp., simulation) estimate of the total population of automata of kind $\langle \boldsymbol{i} \rangle$ in local state j in equilibrium. The approximation error is defined as

$$\text{Relative Density Error} = \frac{\left| O^j\langle \boldsymbol{i} \rangle - S^j\langle \boldsymbol{i} \rangle \right|}{n\langle \boldsymbol{i} \rangle} \times 100.$$

The error statistics across all 500 models are reported in Table 2. As can be seen from the figures, the approximation is highly accurate in all cases, with a maximum error less than 20% and average errors less than 1%. We wish to point out that these results were obtained with relatively small population sizes, where mean-field approximations are known to perform less well. Even under these conditions, ODE

	Original		*New*	
Model	*Max Error*	*Speed-up*	*Max Error*	*Speed-up*
1	20.60	309	8.06	997
2	14.07	183	2.19	642
3	13.47	53	1.32	235
4	11.55	30	2.02	165

Table 3
Approximation errors and speed-ups of the 4 worst-behaving randomly generated models (columns labelled *Original*) against parameterisations with doubled automata populations (columns labelled *New*).

analysis turned out to be significantly faster to run. To study this, we measured the wall-clock execution times of ODE analysis and simulation on a machine equipped with an Intel Core i7 2.66 GHz with 8 GB RAM. Figure 2 shows the speed-up versus the total population, exhibiting a linear trend that grows roughly as 1.8 of the population size. The minimum, average, and maximum speed-ups were found to be ca. 11, 422, and 2000, respectively.

In further experiments we tested the hypothesis that scaling up the population sizes leads to a decrease in the approximation errors, which would be consistent with the general behaviour of mean-field/fluid models. For this study, we considered those models with relative density errors greater than 10% (there were four such models amongst those considered in the original experiment) and repeated their analysis after doubling all the populations of all automata classes $n\langle 1\rangle$, $n\langle 1,1\rangle$, $n\langle 1,2\rangle$ and $n\langle 2\rangle$.

Table 3 shows the maximum errors and the simulation/ODE speed-ups in the original parameterisation against those measured after doubling the automata populations. In all cases, a substantial decrease of the maximum errors can be noted, thus confirming our hypothesis. As for the speed-up, we remark that the ODE system is the same (specifically, it has 8 equations) but increasing automata populations makes simulation more expensive. In these models, we registered runtime slowdown by a factor of over three.

In conclusion, these results suggest that the quality of our fluid approximation is generally satisfactory already with relatively small automata populations, with a tendency to improve significantly with larger populations, where it becomes increasingly more convenient than stochastic simulation.

5 Conclusions

In this paper we introduced an automata-based description of systems of systems, with automata contained in other automata. The dynamics of these automata is described by a set of Markovian transitions (in continuous time), with rates depending on the state of sibling automata, but also on the state of the parent

(downward influence) and of the child nodes (upward influence). Furthermore, causal maps allow us to specify how a transition at a given level can propagate its effect downwards in the containment hierarchy. We also provide a way to flatten a model, and construct a flattened CTMC. Then, we consider how to construct a fluid approximation of this flattened CTMC, and exploit the symmetry of the so-obtained ODEs to lump the fluid state space, reducing the number of equations from one equation per state of each different automata present in the model to one equation per state of each automata class. This reduction is polynomial in the population size and exponential in the nesting level and allows us to approximate efficiently the average of the process. We also discuss the quality of the approximation in a hierarchic model of a computer, showing a very good trade-off between accuracy and computational resources needed. Indeed, in the randomly generated models that we consider the lumped fluid analysis is at least an order of magnitude faster than a simulation based method, even for relatively small populations of computer and processes.

There are several directions for future work. An interesting area of applicability could be biological processes. but for this the model will need to be extended to explicitly treat birth and death events, in order to describe and investigate systems in which the populations of automata are not static throughout the life of the system. Second, we will consider how lumpability can be extended to higher-order moment approximations. Finally, we would like to lift convergence results for Markov population models to our setting of nested automata, for instance by studying which scaling of replica sizes leads to decrease in the approximation error.

Acknowledgement

This work has been partially supported by the EU project QUANTICOL, 600708, by DFG project FEMPA, and by the project FRA-UNITS. The authors thank Stephen Gilmore for his assistance in preparing the paper.

References

[1] Bernardo, M. and R. Gorrieri, *A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time*, Theor. Comput. Sci. **202** (1998), pp. 1–54.

[2] Bortolussi, L., *On the approximation of stochastic concurrent constraint programming by master equation*, ENTCS **220** (2008), pp. 163–180.

[3] Bortolussi, L., J. Hillston, D. Latella and M. Massink, *Continuous approximation of collective systems behaviour: a tutorial*, Performance Evaluation **70** (2013), pp. 317–349.

[4] Buchholz, P., *A class of hierarchical queueing networks and their analysis*, Queueing Systems **15** (1994), pp. 59–80.

[5] Buchholz, P., *An adaptive aggregation/disaggregation algorithm for hierarchical Markovian models*, European Journal of Operational Research **116** (1999), pp. 545–564.

[6] Cerotti, D., M. Gribaudo, A. Bobbio, C. Calafate and P. Manzoni, *A markovian agent model for fire propagation in outdoor environments*, in: *Computer Performance Engineering - 7th European Performance Engineering Workshop, EPEW 2010*, Lecture Notes in Computer Science **6342** (2010), pp. 131–146.

[7] Hayden, R. A. and J. T. Bradley, *A fluid analysis framework for a Markovian process algebra*, Theor. Comput. Sci. **411** (2010), pp. 2260–2297.

[8] Hermanns, H. and M. Rettelbach, *Syntax, semantics, equivalences, and axioms for MTIPP*, in: *Proceedings of PAPM*, Erlangen, 1994, pp. 71–87.

[9] Hillston, J., "A compositional approach to performance modelling," Cambridge University Press, 1996.

[10] Kim, D. S., F. Machida and K. S. Trivedi, *Availability modeling and analysis of a virtualized system*, in: *PRDC*, 2009, pp. 365–371.

[11] Lanus, M., L. Yin and K. S. Trivedi, *Hierarchical composition and aggregation of state-based availability and performability models*, IEEE Transactions on Reliability **52** (2003), pp. 44–52.

[12] Plateau, B. and K. Atif, *Stochastic automata network for modeling parallel systems*, IEEE Trans. Software Eng. **17** (1991), pp. 1093–1108.

[13] Singh, A. and J. Hespanha, *Lognormal moment closures for biochemical reactions*, in: *Proceedings of 45th IEEE Conference on Decision and Control*, 2006.

[14] Tribastone, M., J. Ding, S. Gilmore and J. Hillston, *Fluid rewards for a stochastic process algebra*, IEEE Trans. Software Eng. **38** (2012), pp. 861–874.

[15] Tschaikowski, M. and M. Tribastone, *Exact fluid lumpability for Markovian process algebra*, in: *CONCUR*, 2012, pp. 380–394.

[16] Zhang, X., G. Neglia, J. Kurose and D. Towsley, *Performance modeling of epidemic routing*, Computer Networks **51** (2007), pp. 2867–2891.